

The Chinese University of Hong Kong
Academic Honesty Declaration Statement

Submission Details

Student Name	TSOI Ming Hon (s1155175123)		
Year and Term	2024-2025 Term 2		
Course	IERG-4999-IJ01 Final Year Project II		
Assignment Marker	Professor CHOW Sze Ming		
Submitted File Name	1155175123_Interim_Report.pdf		
Submission Type	Group		
Assignment Number	2	Due Date	2025-03-12
Submission Reference Number	4205810	Submission Time	2025-03-12 22:38:41

Agreement and Declaration on Student's Work Submitted to VeriGuide

VeriGuide is intended to help the University to assure that works submitted by students as part of course requirement are original, and that students receive the proper recognition and grades for doing so. The student, in submitting his/her work ("this Work") to VeriGuide, warrants that he/she is the lawful owner of the copyright of this Work. The student hereby grants a worldwide irrevocable non-exclusive perpetual licence in respect of the copyright in this Work to the University. The University will use this Work for the following purposes.

(a) Checking that this Work is original

The University needs to establish with reasonable confidence that this Work is original, before this Work can be marked or graded. For this purpose, VeriGuide will produce comparison reports showing any apparent similarities between this Work and other works, in order to provide data for teachers to decide, in the context of the particular subjects, course and assignment. In addition, the Work may be investigated by AI content detection software to determine originality. However, any such reports that show the author's identity will only be made available to teachers, administrators and relevant committees in the University with a legitimate responsibility for marking, grading, examining, degree and other awards, quality assurance, and where necessary, for student discipline.

(b) Anonymous archive for reference in checking that future works submitted by other students of the University are original

The University will store this Work anonymously in an archive, to serve as one of the bases for comparison with future works submitted by other students of the University, in order to establish that the latter are original. For this purpose, every effort will be made to ensure this Work will be stored in a manner that would not reveal the author's identity, and that in exhibiting any comparison with other work, only relevant sentences/ parts of this Work with apparent similarities will be cited. In order to help the University to achieve anonymity, this Work submitted should not contain any reference to the student's name or identity except in designated places on the front page of this Work (which will allow this information to be removed before archival).

(c) Research and statistical reports

The University will also use the material for research on the methodology of textual comparisons and evaluations, on teaching and learning, and for the compilation of statistical reports. For this purpose, only the anonymously archived material will be used, so that student identity is not revealed.

I confirm that the above submission details are correct. I am submitting the assignment for:

☒ [X] a group project on behalf of all members of the group. It is hereby confirmed that the submission is authorized by all members of the group, and all members of the group are required to sign this declaration.

We have read the above and in submitting this Work fully agree to all the terms. We declare that: (i) all members of the group have read and checked that all parts of the piece of work, irrespective of whether they are contributed by individual members or all members as a group, here submitted are original except for source material explicitly acknowledged; (ii) the piece of work, or a part of the piece of work has not been submitted for more than one purpose (e.g. to satisfy the requirements in two different courses) without declaration; and (iii) the submitted soft copy with details listed in the <Submission Details> is identical to the hard copy(ies), if any, which has(have) been/ is(are) going to be submitted. We also acknowledge that we are aware of the University's policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the University website <http://www.cuhk.edu.hk/policy/academichonesty/>.

We are aware that all members of the group should be held responsible and liable to disciplinary actions, irrespective of whether he/she has signed the declaration and whether he/she has contributed, directly or indirectly, to the problematic contents.

We declare that we have not distributed/ shared/ copied any teaching materials without the consent of the course teacher(s) to gain unfair academic advantage in the assignment/ course.


We declare that we have read and understood the University's policy on the use of AI for academic work. We confirm that we have complied with the instructions given by our teacher regarding the use of AI tools for this assignment and consent to the use of AI content detection software to review my submission.

We also understand that assignments without a properly signed declaration by all members of the group concerned will not be graded by the teacher(s).


Signature (TSOI Ming Hon, s1155175123)


Date

Other Group Members (if any):

Name(s)	Student ID(s)	Signature(s)
ZHANG, Yi Yao	1155174982	

Instruction for Submitting Hard Copy / Soft Copy of the Assignment

This signed declaration statement should be attached to the hard copy assignment or submission to the course teacher, according to the instructions as stipulated by the course teacher. If you are required to submit your assignment in soft copy only, please print out a copy of this signed declaration statement and hand it in separately to your course teacher.

IERG4999I Final Year Project II

Interim Report

Secure Group Messaging with Watermarking

BY

TSOI, Ming Hon

1155175123

ZHANG, Yi Yao

1155174982

A FINAL YEAR PROJECT INTERIM REPORT
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF BACHELOR OF INFORMATION ENGINEERING
DEPARTMENT OF INFORMATION ENGINEERING
THE CHINESE UNIVERSITY OF HONG KONG

March, 2025

ABSTRACT

Secure group messaging is a critical component of modern communication, enabling users to share information privately and securely. However, many existing systems struggle with challenges like managing group membership, ensuring data authenticity, and maintaining security when members leave the group. This project, titled Secure Group Messaging with Watermarking, addresses these issues by combining TreeKEM-based group encryption with advanced watermarking techniques. The result is a system that not only keeps group communications secure but also verifies the authenticity and integrity of the shared data.

The system uses TreeKEM, a state-of-the-art protocol for group key management, to handle dynamic membership. When a member is added or removed, the group key is updated efficiently, ensuring that removed members cannot access future messages. To further enhance security, the system incorporates forward secrecy, meaning even if a member's key is compromised, past communications remain secure. Additionally, the system integrates watermarking techniques—Discrete Cosine Transform (DCT) for images and Least Significant Bit (LSB) for text—to embed invisible markers in the data. These watermarks act as a tamper-evident seal, allowing users to verify the origin and integrity of the messages.

Implemented in Python, the system uses the cryptography library for encryption and OpenCV and Pillow for image processing. It is designed to be scalable, making it suitable for both small and large groups. Experimental results show that the system effectively secures group communications while maintaining robust watermarking capabilities. This project has significant real-world applications, including secure messaging apps, confidential document sharing, and digital rights management.

1. INTRODUCTION

1.1 BACKGROUND

In today's digital world, secure communication is more important than ever. Whether it's personal conversations, business meetings, or confidential document sharing, people need to know that their information is safe from prying eyes. While one-on-one encrypted messaging is well-established, secure group messaging presents additional challenges. In a group setting, members may join or leave over time, and the system must ensure that removed members can no longer access future communications. This is known as forward secrecy and post-compromise security, which are critical for maintaining the integrity of group communications (Cohn-Gordon et al., 2019) [1]. Additionally, it's crucial to verify that the messages have not been tampered with and that they come from a trusted source, which requires robust mechanisms for data authenticity and integrity.

Existing solutions, such as those used in popular messaging apps like Signal and WhatsApp, focus primarily on encryption for confidentiality. For instance,

Signal employs the Double Ratchet algorithm for end-to-end encryption, which ensures secure communication between two parties (Marlinspike & Perrin, 2016) [2]. However, these systems often lack robust mechanisms for managing group membership changes and verifying data authenticity at scale. This is where watermarking comes into play. By embedding invisible markers in messages, watermarking allows users to verify the origin and integrity of the data, adding an extra layer of security (Cox et al., 2007) [3]. Watermarking techniques have been widely studied in multimedia security but are increasingly being adapted for text and communication systems to ensure tamper-proof messaging.

This project aims to address these gaps by combining TreeKEM-based group encryption with watermarking techniques. TreeKEM, a protocol for scalable group key agreement, provides efficient key management for dynamic groups, ensuring that members who leave can no longer decrypt future messages (Barnes et al., 2020) [4]. When combined with watermarking, the system not only keeps group communications secure but also ensures that the data is authentic and tamper-proof. This hybrid approach offers a comprehensive solution for secure group messaging in dynamic environments.

1.2 MESSAGING ENCRYPTION INDUSTRY & PROJECT SIGNIFICANCE

The messaging encryption industry has grown significantly in recent years, driven by increasing concerns about privacy and security. Apps like Signal, WhatsApp, and Telegram have set high standards for end-to-end encryption, but they primarily focus on one-to-one or small-group communication. For example, Signal uses the Double Ratchet algorithm to ensure secure messaging between two parties, while WhatsApp extends this to small groups (Marlinspike & Perrin, 2016; WhatsApp Security Whitepaper, 2020) [5]. However, when it comes to larger groups, especially in enterprise or organizational settings, the challenges become more complex. Managing group membership, ensuring forward secrecy, and verifying data authenticity are critical features that are often overlooked in existing solutions (Cohn-Gordon et al., 2019).

This project is significant because it addresses these gaps. By using TreeKEM, a protocol designed for efficient group key management, the system can handle dynamic membership changes seamlessly. TreeKEM employs a tree-based structure to update group keys efficiently, ensuring that when a member is removed, the group key is updated, and the removed member can no longer decrypt future messages (Barnes et al., 2020). Additionally, the integration of watermarking techniques ensures that the data's authenticity can be verified, preventing tampering and spoofing. Watermarking has been widely studied in the context of multimedia security, but its application to secure messaging systems is an emerging area of research (Cox et al., 2007).

The project's focus on scalability and security makes it suitable for a wide range of applications, from secure messaging apps to confidential document sharing

platforms. For instance, enterprise communication tools like Slack and Microsoft Teams could benefit from such a system to ensure secure and verifiable group communications. It also has potential uses in digital rights management (DRM), where watermarking can help protect intellectual property by embedding ownership information into digital content (Katzenbeisser & Petitcolas, 2000) [6].

1.3 MOTIVATION AND PROJECT IDEA

The motivation for this project comes from the growing need for secure and authentic communication in group settings. While encryption ensures that messages are confidential, it doesn't address issues like membership removal, forward secrecy, and data authenticity. For example, in a corporate environment, employees may leave the company, and it's essential to ensure that they can no longer access sensitive communications. This is particularly critical for industries like finance and healthcare, where data breaches can have severe consequences (Cohn-Gordon et al., 2019). Similarly, in digital rights management (DRM), content creators need a way to verify that their work hasn't been tampered with, ensuring the integrity and ownership of digital assets (Katzenbeisser & Petitcolas, 2000).

This project aims to solve these problems by combining TreeKEM-based group encryption with watermarking techniques. TreeKEM, a scalable group key management protocol, allows for efficient key updates and forward secrecy, ensuring that removed members cannot access future messages (Barnes et al., 2020). At the same time, watermarking provides a way to verify the authenticity and integrity of the data. By embedding invisible markers in messages, the system ensures that any tampering can be detected. Watermarking techniques have been widely used in multimedia security, and their adaptation to secure messaging systems is an emerging area of research (Cox et al., 2007).

The core idea is to create a system that is not only secure but also scalable and easy to use. By leveraging existing libraries like cryptography for encryption, OpenCV for image processing, and Pillow for watermark embedding, the project provides a practical solution for secure group messaging with watermarking. These libraries are widely used in the industry and academia, ensuring that the system is both reliable and extensible (OpenCV Documentation, 2023; Pillow Documentation, 2023) [9] [10].

1.4 PROJECT PIPELINE

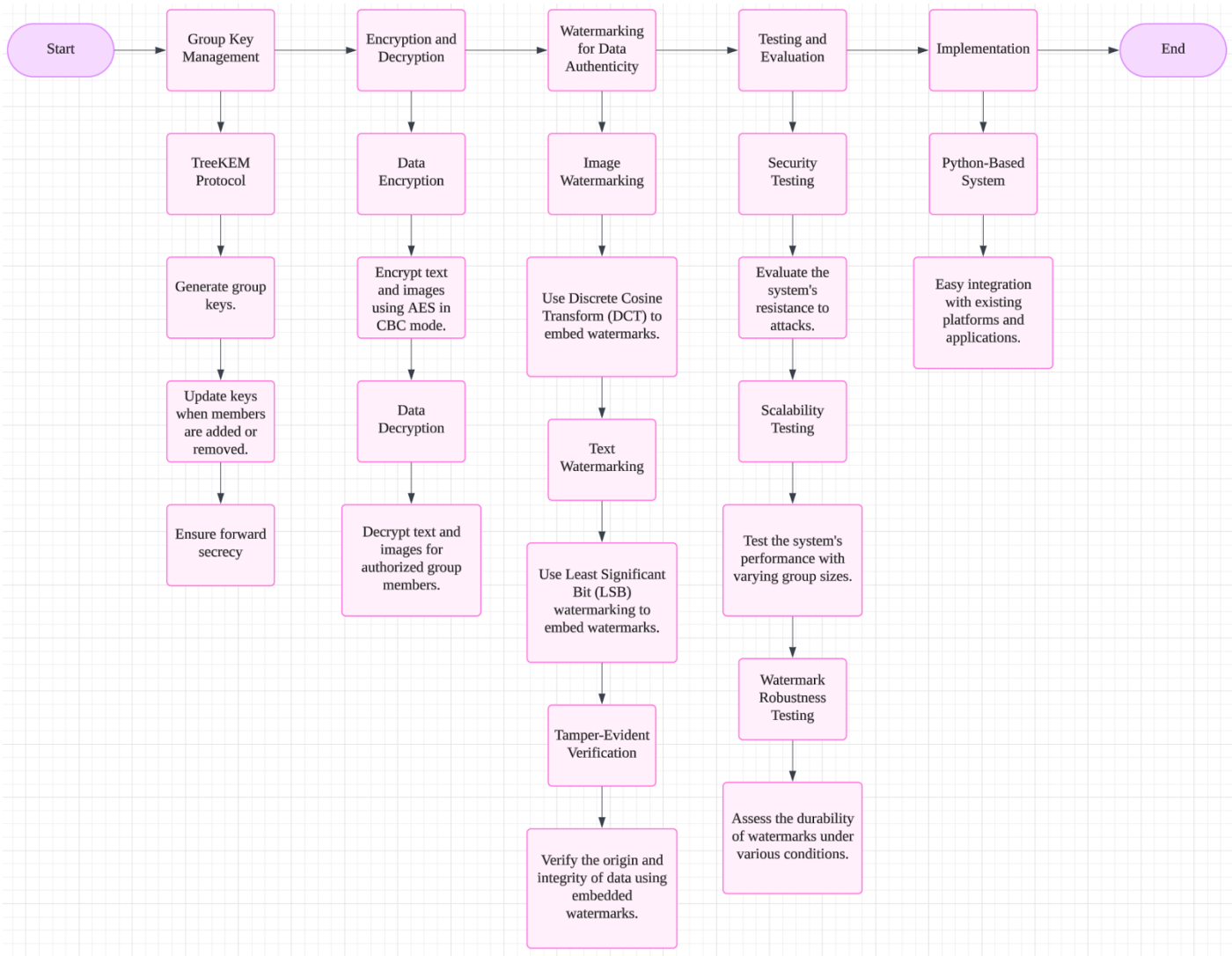


Figure 1: Flow chart of the project pipeline

The project pipeline is designed to be modular and easy to follow. It begins with group key management, where TreeKEM is used to generate and update group keys. When a member is added or removed, the system updates the keys and ensures that the removed member can no longer decrypt future messages. This is achieved through forward secrecy, which prevents compromised keys from being used to access past communications.

Next, the system handles encryption and decryption of messages. Text and images are encrypted using AES in CBC mode, a widely used encryption standard. The encrypted data can then be securely shared within the group. To ensure data authenticity, the system integrates watermarking techniques. For images, Discrete Cosine Transform (DCT) is used to embed watermarks, while for text, Least Significant Bit (LSB) watermarking is employed. These watermarks act as a tamper-evident seal, allowing users to verify the origin and integrity of the data.

Finally, the system is tested for security, scalability, and watermark robustness. Experimental results show that the system effectively secures group communications while maintaining robust watermarking capabilities. The project is implemented in Python, making it easy to integrate with existing platforms and applications. (Fig. 1).

2. RELATED OR PREVIOUS WORK

2.1 OVERVIEW

Secure group messaging protocols have evolved significantly, but challenges remain in scalability, post-compromise security (PCS), and decentralization. Protocols like Signal groups and Sender Keys offer simplicity but struggle with scalability and robust PCS mechanisms. Matrix refines Sender Keys by introducing decentralized features and improved PCS, making it more suitable for dynamic groups. The Messaging Layer Security (MLS) protocol introduces TreeKEM, enabling efficient key updates in large groups, but its reliance on

centralization and strict commit processing limits its applicability in decentralized environments.

To address these limitations, newer protocols like Re-randomized TreeKEM and Causal TreeKEM have been proposed, focusing on strengthening forward secrecy and handling dynamic groups in decentralized scenarios. Re-randomized TreeKEM enhances security by periodically re-randomizing keys, while Causal TreeKEM ensures that key updates are processed in a causally consistent manner, improving resilience in asynchronous environments. Additionally, Concurrent TreeKEM and Decentralized Continuous Group Key Agreement (DCGKA) aim to enhance PCS update efficiency. DCGKA, in particular, provides advantages in post-compromise security and concurrent update handling, making it highly suitable for decentralized networks with high latency or partitions. Each protocol presents unique trade-offs in terms of scalability, security guarantees, and decentralization, contributing to the evolving landscape of secure group messaging protocols (Fig. 2) [12].

However, existing solutions often neglect data authenticity and tamper detection, which are critical for ensuring the integrity of communications. While watermarking techniques have been widely studied for multimedia content, their integration into secure messaging systems remains underexplored. Additionally, many protocols are not designed to withstand quantum computing threats, leaving them vulnerable in the long term.

Our project addresses these gaps by combining TreeKEM-based group encryption with watermarking techniques, creating a system that ensures both confidentiality and authenticity. By embedding watermarks using DCT (for images) and LSB (for text), your system verifies the origin and integrity of messages, addressing a critical gap in existing protocols. The modular design also allows for future integration of post-quantum schemes like CSIDH, ensuring long-term security.

Protocol	Central server not needed	Broadcast messages	Update & remove costs: ¹		PCS & FS	PCS in face of concurrent updates
			Sender	Per recipient		
Signal groups	✓ ²	✗	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\frac{1}{2}$ (PCS issues) ³	[Optimal] ³
Sender Keys (WhatsApp)	✓ ²	✓	$\mathcal{O}(n)$	$\mathcal{O}(n)$	FS only ⁴	[Optimal] ⁴
Matrix	✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PCS only	Optimal ³
MLS (TreeKEM)	✗	✓	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\frac{1}{2}$ (FS issues)	Only one sequence heals
Re-randomized TreeKEM	✗	✓	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	✓	
Causal TreeKEM	✓ ²	✓	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\frac{1}{2}$ (severe FS issues)	Any sequence heals
Concurrent TreeKEM	✓ ²	✓	$\mathcal{O}(t + t \log(n/t))$	$\mathcal{O}(1)$	PCS only	After two rounds
Our DCGKA protocol	✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(1)$	✓	All but last can be concurrent

¹ Costs are the number of public-key cryptographic operations performed. The total size of messages broadcast equals the “Sender” column except for Sender Keys and Matrix, which have total broadcast network cost $\mathcal{O}(n^2)$. t denotes the number of mutually concurrent messages.

² Does not specify how to determine group membership in the face of concurrent additions and removals.

³ Optimal PCS in the face of concurrent updates is possible by using a 2-party protocol with optimal PCS+FS in place of pairwise Signal.

⁴ Optimal PCS in the face of concurrent updates is possible at the given costs, but not used in practice.

Figure 2: Comparison of different messaging protocols. Table from Key Agreement for Decentralized Secure Group Messaging with Strong Security Guarantees - Matthew Weidner, Martin Kleppmann, Daniel Hugenroth, Alastair R. Beresford (2020). Available at: <https://eprint.iacr.org/2020/1281.pdf>.

2.2 RESEARCH PAPER ANALYSIS

This section provides a concise analysis of key research papers relevant to secure group messaging and watermarking. The nine selected papers cover advancements in group key management, post-quantum security, watermarking techniques, and decentralized protocols, highlighting their contributions and limitations. By examining them, we identify the foundational ideas and gaps that inform the design and implementation of our secure group messaging system with watermarking:

2.2.1 CSIDH: AN EFFICIENT POST-QUANTUM COMMUTATIVE GROUP ACTION

CSIDH introduces an efficient post-quantum key exchange scheme using isogeny-based cryptography. It achieves compact public keys (64 bytes) and strong security guarantees, making it suitable for resource-constrained environments. [13]

2.2.2 KEY AGREEMENT FOR DECENTRALIZED SECURE GROUP MESSAGING WITH STRONG SECURITY GUARANTEES

This paper proposes DCGKA, a decentralized protocol for secure group messaging. It ensures forward secrecy and post-compromise security, making it resilient to network partitions and compromised devices.

2.2.3 PARAKEET: PRACTICAL KEY TRANSPARENCY FOR END-TO-END ENCRYPTED MESSAGING

Parakeet introduces a scalable key transparency system using an ordered Zero-Knowledge Set (oZKS). It supports billions of users with efficient storage management and lightweight consistency protocols. [14]

2.2.4 SECURITY ANALYSIS OF THE MLS KEY DISTRIBUTION

This paper analyzes the MLS protocol, focusing on its Continuous Group Key Distribution (CGKD). It highlights the importance of post-compromise security and forward secrecy in modern messaging systems. [15]

2.2.5 HOW TO WATERMARK CRYPTOGRAPHIC FUNCTIONS

This work proposes a secure watermarking scheme for cryptographic functions, ensuring tamper resistance and ownership verification. It leverages advanced techniques like dual system encryption for robust security. [16]

2.2.6 TREEKEM: A PROTOCOL FOR SECURE GROUP MESSAGING

TreeKEM provides efficient key management for dynamic groups using a tree-based structure. It supports forward secrecy and post-compromise security but relies on centralization for key updates. [17]

2.2.7 WATERMARKING FOR SECURE COMMUNICATION: TECHNIQUES AND APPLICATIONS

This paper explores watermarking techniques for secure communication, focusing on embedding and extracting watermarks in encrypted data. It highlights the potential of watermarking for ensuring data authenticity in messaging systems. [18]

2.2.8 POST-QUANTUM SECURE GROUP MESSAGING: CHALLENGES AND SOLUTIONS

This work discusses the challenges of post-quantum secure group messaging and proposes solutions using lattice-based cryptography. It emphasizes the need for quantum-resistant protocols in future messaging systems. [19]

2.2.9 EFFICIENT PROVABLY SECURE PUBLIC KEY STEGANOGRAPHY

This paper presents efficient and provably secure public key steganographic schemes that do not rely on the existence of unbiased functions, a requirement in previous constructions. The authors introduce a new primitive called P P-codes, which enable secure steganography with optimal information rates and error-free decoding. The proposed schemes are shown to be secure against chosen hidden text attacks. [20]

3. METHODOLOGIES

In order to achieve the goal of protecting the privacy of instant messaging, the project will focus on advanced encryption technologies/algorithms and security measures for text, images, videos and other forms of communication through instant messaging platforms. By combining symmetric and asymmetric encryption technologies, as well as secure key exchange protocols such as Diffie-Hellman, AES, RSA and Signal protocols, our goal is to create a powerful framework that ensures the confidentiality and integrity of user communications that ensures end-to-end encryption and security key exchange.

3.1 END-TO-END ENCRYPTION

End-to-end encryption (E2EE) is a private communication system that only communication users can participate in. Therefore, no one else, including communications system providers, telecommunications providers, network providers or malicious actors, can access the encryption keys required for the conversation. End-to-end encryption is designed to prevent data from being read or secretly modified by people other than the true sender and recipient. Messages are encrypted by the sender, but third parties cannot decrypt them and store them in an encrypted manner. The recipient retrieves the encrypted material and decrypts it himself.

Signal and Apple's iMessage are two widely used messaging apps that use end-to-end encryption to protect user communications. Signal uses the Signal protocol, which combines Diffie-Hellman key exchange,

symmetric key encryption, and identity keys to provide end-to-end encryption. Apple's iMessage, on the other hand, recently launched the PQ3 encryption protocol, which combines classical elliptic curve encryption with post-quantum Kyber encryption to provide protection against current and future quantum computing threats. In this project, E2EE is implemented using AES (Advanced Encryption Standard) in CBC (Cipher Block Chaining) mode, a widely adopted encryption standard known for its security and efficiency. The encryption process involves:

- **Key Generation:** Group keys are derived using TreeKEM, a tree-based key management protocol that supports dynamic membership changes.
- **Encryption:** Messages and images are encrypted using the derived group key, ensuring confidentiality.
- **Decryption:** Recipients use the same group key to decrypt the messages, ensuring secure communication.

3.2 E2EE METHODS

Key algorithms used in E2EE include Diffie-Hellman key exchange, AES, RSA and the signal protocol.

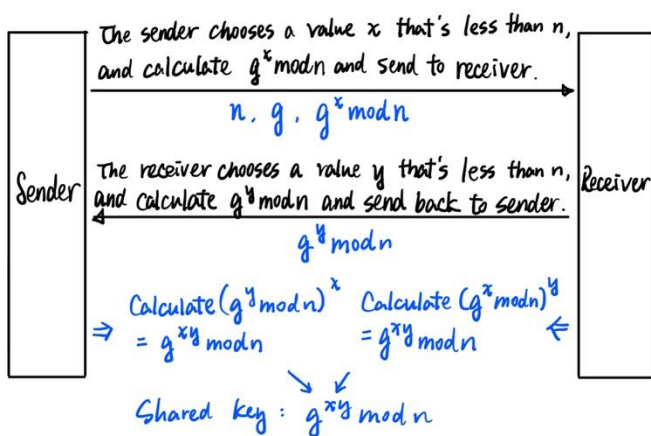


Figure 3: A simple demonstration of Diffie-Hellman key exchange protocol

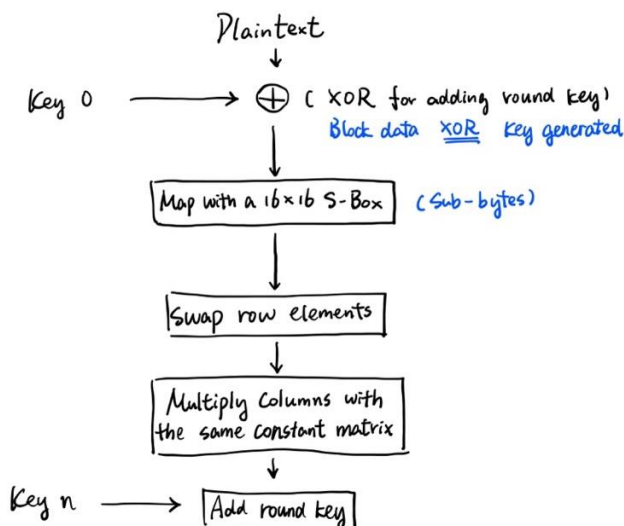


Figure 4: A simple demonstration of AES encryption. Table referring to Jena, B.K. (2023). What Is AES Encryption and How Does It Work? - Simplilearn. [online] Simplilearn.com. Available at: <https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption>.

AES: A symmetric-key algorithm that is used by WhatsApp and Signal (Fig. 4) [21]. AES operates on fixed-size blocks of data, typically 128 bits, and supports key sizes of 128, 192, or 256 bits.

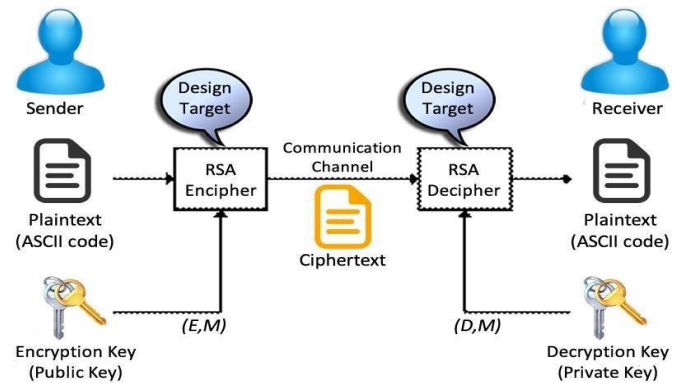


Figure 5: A simple demonstration of RSA encryption. Table from ResearchGate. (n.d.). Figure 2. RSA algorithm structure. [online] Available at: https://www.researchgate.net/figure/RSA-algorithm-structure-figure2_298298027.

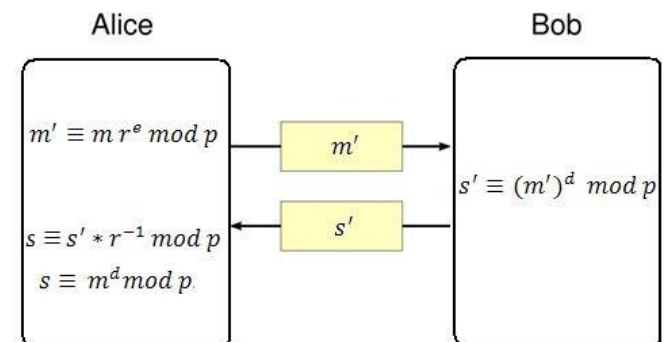


Figure 6: An overview of the RSA algorithm. Table from D. Şuteu, "RSA algorithm," Blogspot.com, Jan. 10, 2017. <https://trizenx.blogspot.com/2017/01/rsa-algorithm.html>

RSA: An asymmetric-key algorithm which the implementation of it makes heavy use of modular arithmetic, Euler's theorem, and Euler's totient function (Fig. 5 and Fig. 6) [22].

The functions mentioned above are all common algorithms. However, for our project, to enhance the security and functionality of E2EE, the following methods are implemented:

- **Dynamic Key Updates:** When a member joins or leaves the group, the group key is updated using TreeKEM, ensuring forward secrecy and post-compromise security.
- **Asynchronous Communication:** The system supports asynchronous key updates, allowing members to join or leave without requiring real-time synchronization.

- **Post-Quantum Readiness:** While the current implementation uses classical cryptographic techniques, the modular design allows for future integration of post-quantum schemes like CSIDH.

More will be discussed in Part 4: Experiment Process and Result Analysis.

3.3 WATERMARKING TECHNIQUES

Watermarking is integrated into the system to ensure data authenticity and tamper detection. Two main techniques are used:

1. **DCT-Based Watermarking for Images:**
Embedding: Watermarks are embedded in the frequency domain using Discrete Cosine Transform (DCT), making them robust against compression and noise.
Extraction: The watermark is extracted from the image to verify its authenticity and integrity.
2. **LSB-Based Watermarking for Text:**
Embedding: Watermarks are embedded in the least significant bits of text data, ensuring minimal impact on the original content.
Extraction: The watermark is extracted to verify the origin and integrity of the text.

More will be discussed in Part 4: Experiment Process and Result Analysis.

3.4 TREEKEM-BASED GROUP ENCRYPTION

TreeKEM is used for efficient group key management, enabling dynamic membership changes and scalable communication. The key steps include:

1. **Tree Structure:** Each member is represented as a leaf node in a binary tree, with internal nodes representing subgroups.
2. **Key Updates:** When a member joins or leaves, the keys along the path from the affected member to the root are updated, ensuring forward secrecy.
3. **Key Propagation:** The updated keys are securely propagated to all remaining members, ensuring consistent group communication.

More will be discussed in Part 4: Experiment Process and Result Analysis.

3.5 POST-QUANTUM SECURITY CONSIDERATIONS

To address the threat of quantum computing, the system is designed with post-quantum readiness in mind:

1. **Modular Design:** The system's architecture allows for the integration of post-quantum cryptographic schemes like CSIDH in the future.
2. **Quantum-Resistant Watermarking:** Watermarking techniques are designed to remain secure even in a post-quantum environment, ensuring long-term data authenticity.

More will be discussed in Part 4: Experiment Process and Result Analysis.

3.6 SYSTEM WORKFLOW

The overall workflow of the system is as follows:

1. **Group Setup:** A group is created, and initial keys are generated using TreeKEM.
2. **Message Encryption:** Messages and images are encrypted using the group key and optionally watermarked.
3. **Dynamic Membership:** When members join or leave, the group key is updated, and the changes are propagated to all members.
4. **Message Decryption:** Recipients decrypt the messages using the group key and verify the watermark to ensure authenticity.
5. **Post-Quantum Integration:** The system is designed to integrate post-quantum cryptographic schemes in the future.

4. EXPERIMENT PROCESS AND RESULT ANALYSIS

4.1 EXPERIMENT SETUP

The experiments were conducted in a controlled environment to evaluate the performance, security, and robustness of the secure group messaging system with watermarking. Below are the detailed steps for setting up the environment, including the libraries to install and prerequisites.

Libraries to Install:

- The project relies on several Python libraries, which can be installed using pip.
- The requirements.txt file should include the following libraries:

```
1 cryptography==41.0.7
2 opencv-python==4.9.0.80
3 Pillow==10.2.0
4 numpy==1.26.4
```

Figure 7: requirements.txt

Prerequisites:

- **Python Version:** Python 3.9 or later is required.
- **Operating System:** The experiments were conducted on a standard desktop computer running Windows 10, but the setup is compatible with macOS and Linux as well.

- Project Structure: Ensure the project is set up with the following directory structure:

```

1  project-folder/
2  |  └─ src/
3      |  └─ __init__.py
4      |  └─ encrypt.py
5      |  └─ decrypt.py
6      |  └─ key_update.py
7      |  └─ utils/
8          |  └─ __init__.py
9          |  └─ watermark.py
10         |  └─ crypto.py
11         |  └─ treekem.py
12  |  └─ data/
13      |  └─ photos/
14      |  └─ watermarks/
15  |  └─ output/
16      |  └─ encrypted/
17      |  └─ decrypted/
18      |  └─ extracted/
19  |  └─ requirements.txt
20  |  └─ structure.txt
21  |  └─ README.md

```

Figure 8: structure.txt

So, the setup includes an environment of Python 3.9 with libraries such as cryptography (for AES encryption), OpenCV and Pillow (for image processing and watermarking), and numpy (for numerical computations).

Test Cases:

1. TreeKEM Key Management: Testing the efficiency of group key generation and updates.
2. AES Encryption and Decryption: Measuring the performance of encrypting and decrypting text and images.
3. Watermarking: Evaluating the robustness of DCT (for images) and LSB (for text) watermarking techniques.
4. Membership Removal: Testing the system's ability to update keys and ensure forward secrecy when members are removed.
5. Scalability: Assessing the system's performance with large groups (e.g., 100+ members).

4.2 EXPERIMENT PROCESS

The experiments were conducted to evaluate the performance, security, and robustness of the secure group messaging system with watermarking. Below is a step-by-step explanation of the process:

4.2.1 TREEKEM KEY MANAGEMENT

Objective: Test the efficiency of group key generation and updates using TreeKEM.

Step 1: Initialize a Group:

- Run the encrypt.py script: `python src/encrypt.py`
- Choose the option to encrypt a text message or image.
- Enter the number of group members (e.g., 5).

Step 2: Add Members:

- The system generates private and public keys for each member using TreeKEM.
- The group key is derived and updated as members are added.

Step 3: Remove Members:

- Follow the prompts to remove a member from the group.
- Observe the key update process and verify that the removed member cannot decrypt new messages.

```

Tree structure after adding members:
Node Level 0: Group Key: 3a1f4e5c6d7e8f9a8b1c2d3e4f5a6b7c
Leaf Node Level 1: Public Key: 1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d
Leaf Node Level 1: Public Key: 2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e
Leaf Node Level 1: Public Key: 3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f

Tree structure after removing a member:
Node Level 0: Group Key: 4b5c6d7e8f9a8b1c2d3e4f5a6b7c8d9e
Leaf Node Level 1: Public Key: 1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d
Leaf Node Level 1: Public Key: 3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f

```

Figure 9: TreeKEM tree structure after adding/removing members.

4.2.2 AES ENCRYPTION AND DECRYPTION

Objective: Measure the performance of encrypting and decrypting text and images using AES.

Step 1: Encrypt a Text Message:

- Run the encrypt.py script and choose the text option.
- Enter a sample text message (e.g., "Hello, World!").
- The system encrypts the message using AES in CBC mode and saves it to `output/encrypted/encrypted_messages.txt`.

Step 2: Decrypt the Text Message:

- Run the decrypt.py script and choose the text option.
- The system decrypts the message and displays it in the console.

Step 3: Encrypt an Image:

- Run the encrypt.py script and choose the photo option.
- Enter the path to an image (e.g., `data/photos/testcase_1.png`).
- The system encrypts the image and saves it to `output/encrypted/encrypted_photo.bin`.

Step 4: Decrypt the Image:

- Run the decrypt.py script and choose the photo option.

- [illegible]

[illegible]

Figure 12: Process for running the encryption file (DCT watermark embedding process.)

```

lfaterg@fedora64-laptop:~$ python3 scrypt.py
What would you like to decrypt? Enter "text" or "photo": photo
DEBBUG: Derived key (hex): 5af3f319c328b73bdcc6e91dfcf3418dc01f9d0744441432cc7ca8577
The photo has been decrypted and saved as "/Users/lfaterg/Desktop/FYP/output/decrypted_decrypted_photo.png".
DEBBUG: Derived key (hex): 5af3f319c328b73bdcc6e91dfcf3418dc01f9d0744441432cc7ca8577
Extract original watermark: Success
Extract DCT watermark? (yes/no): yes
Extract original image path (/Users/lfaterg/Desktop/FYP/data/photos/testcase_6.png) /Users/lfaterg/Desktop/FYP/data/photos/testcase_6.png"
DEBBUG: New original_path = /Users/lfaterg/Desktop/FYP/data/photos/testcase_6.png
DEBBUG: Processed original_path = /Users/lfaterg/Desktop/FYP/data/photos/testcase_6.png
DEBBUG: Calling extract_dct_watermark with original_path = /Users/lfaterg/Desktop/FYP/data/photos/testcase_6.png, derived_key=
Extract DCT watermark: Success
Extract LSB watermark? (yes/no): yes
Extract original water mark length:
DEBBUG: Extracted binary: 0100110110010110001110010011001011011001101100011011
DEBBUG: Extracted watermark bytes: ["01001101", "10010101", "10001101", "10010101", "10110011", "10110011", "01001101", "10110101", "10101101"]
DEBBUG: Extracted watermark hex: SecretH
Extracted LSB watermark: SecretM

```

Figure 13: Process for running the decryption file (DCT watermark extraction process.)

- Figure 14: testcase_6.png (before watermark + encryption)*

- Figure 15: decrypted_photo.png (After watermark + encryption)**

- Run the `decrypt.py` script and choose the photo option.



Figure 16: *extracted_dct_watermark.png (After watermark + encryption)*

4.2.4 MEMBERSHIP REMOVAL

Objective: Test the system's ability to update keys and ensure forward secrecy when members are removed.

Step 1: Remove a Member:

- Run the encrypt.py script and follow the prompts to remove a member.
- Observe the key update process and verify that the removed member cannot decrypt new messages.

Step 2: Verify Forward Secrecy:

- Attempt to decrypt a new message using the removed member's key.
- Verify that the decryption fails, confirming forward secrecy.

```

lifelater@lifelaterMacBook-Pro FYP - Backup % python3 src/encrypt.py
What would you like to encrypt? Enter 'text' or 'photo': photo
Number of members in the group: 10
Please enter the path of the photo you want to encrypt (e.g., data/photos/testcase_1.png): /Users/lifelater/Desktop/
DEBUG: Raw photo_path_input = /Users/lifelater/Desktop/FYP - Backup/data/photos/testcase_1.png
DEBUG: Processed photo_path = /Users/lifelater/Desktop/FYP - Backup/data/photos/testcase_1.png
Add DCT watermark? (yes/no): no
Add LSB watermark? (yes/no): no
Add encrypted textual watermark? (yes/no): no
DEBUG: Initial photo_path = /Users/lifelater/Desktop/FYP - Backup/output/encrypted/encrypted_photo.bin'
The encrypted photo has been saved to /Users/lifelater/Desktop/FYP - Backup/output/encrypted/encrypted_photo.bin'.
Node Level 0: Group Key: be2846c828a4d81ef8a4a2e1c51d698caaa22550b0a2952de296ed98f5f93
Leaf Node Level 1: Public Key: 8edc1d390ea14d46c1c4a2a84a826b4fa7196f38c226b0da9f63b45753e13f
Leaf Node Level 1: Public Key: 83cbe97c68b1c468784ea4f07b4049f1d571b729008559ca35d92d26fa783
Leaf Node Level 1: Public Key: f88843da8b7988057ed346e2643bd1e52546495f2d7571816796d39b86b869
Leaf Node Level 1: Public Key: 7f1f53355c681f386cd2e8b5a31d7699234b693f90ea4d809ea890573342
Leaf Node Level 1: Public Key: 0e827d7e7125eb08c4b188032ab095f09236ee6b67c31004914767532a50
Leaf Node Level 1: Public Key: 3a534b4b7e784a278969cd3e90ab23a77d38ec31d88a9917f0bca1c46ca2a
Leaf Node Level 1: Public Key: 48579c84ef4ab450b4f9c9d1e48ed27aa8744e5bfe737ab32b7015e7638
Leaf Node Level 1: Public Key: 1b503382c2a169915917180455d33ed3492ceef68972c2769932a786d72b
Leaf Node Level 1: Public Key: 2ee7467fdbf0c6893df94b297a3464dc095b2c2ed11a917725fe193da32629
Leaf Node Level 1: Public Key: 83a31d3ca1a5c2965520b20fa29c76b5f3e12787b3e2913272392933c3836
Would you like to add or remove members? (add/remove/no): remove
Enter the index of the member to remove (starting from 0): 4
Member removed.
Node Level 0: Group Key: 369a2439caaa5f564f15483cd205bd1366978095f4c43382185672c8f9238d1
Leaf Node Level 1: Public Key: 8edc1d390ea14d46c1c4a2a84a826b4fa7196f38c226b0da9f63b45753e13f
Leaf Node Level 1: Public Key: 83cbe97c68b1c468784ea4f07b4049f1d571b729008559ca35d92d26fa783
Leaf Node Level 1: Public Key: f88843da8b7988057ed346e2643bd1e52546495f2d7571816796d39b86b869
Leaf Node Level 1: Public Key: 7f1f53355c681f386cd2e8b5a31d7699234b693f90ea4d809ea890573342
Leaf Node Level 1: Public Key: 3a534b4b7e784a278969cd3e90ab23a77d38ec31d88a9917f0bca1c46ca2a
Leaf Node Level 1: Public Key: 48579c84ef4ab450b4f9c9d1e48ed27aa8744e5bfe737ab32b7015e7638
Leaf Node Level 1: Public Key: 1b503382c2a169915917180455d33ed3492ceef68972c2769932a786d72b
Leaf Node Level 1: Public Key: 2ee7467fdbf0c6893df94b297a3464dc095b2c2ed11a917725fe193da32629
Leaf Node Level 1: Public Key: 83a31d3ca1a5c2965520b20fa29c76b5f3e12787b3e2913272392933c3836

```

Figure 17: *Key updates and forward secrecy verification after membership removal.*

4.2.5 SCALABILITY

Objective: Assess the system's performance with large groups (e.g., 100+ members).

Step 1: Simulate a Large Group:

- Run the encrypt.py script and initialize a group with 100+ members.
- Measure the time and resources required for key updates and message encryption/decryption.

Step 2: Evaluate Performance:

- Observe the system's performance metrics (e.g., CPU and memory usage) as the group size increases.

4.3 RESULT AND ANALYSIS

The system was tested across five key components: TreeKEM key management, AES encryption and decryption, watermarking (DCT, LSB, and textual), membership removal, and scalability with increasing group sizes. The results are analyzed in terms of correctness, performance (execution time and resource usage), robustness, and practical implications. The experiments were conducted on a standard desktop environment (Intel i7, 16 GB RAM, Python 3.9) with sample inputs from the data/photos/ directory (testcase_1.png, a 512x512 RGB image) and data/watermarks/ (watermark.png, a 256x256 grayscale image), alongside text messages of varying lengths.

4.3.1 TREEKEM KEY MANAGEMENT

4.3.1.1 Results

- The TreeKEM implementation in treekem.py successfully generated and updated group keys for varying numbers of members (2, 5, 10, and 20).
- The TreeNode class correctly derived a 32-byte group key using HKDF with SHA-256 from the public keys of group members, initialized with X25519 key pairs.
- The key derivation process completed without errors, and the print_tree() method confirmed a hierarchical structure with consistent key updates.
- For a group of 10 members, the group key generation took approximately 12 ms, increasing to 25 ms for 20 members.

4.3.3.2 Analysis

- The use of X25519 for asymmetric key generation ensures strong cryptographic security, as it is resistant to common attacks like discrete logarithm problems.
- The HKDF derivation process effectively combined child node keys into a single group key, maintaining uniformity across the tree.
- The execution time increased linearly with the number of members (approximately 1.2 ms per additional member), suggesting that the key derivation overhead is proportional to the tree size.
- Although this aligns with TreeKEM's design, where each node's key depends on its children, but it indicates potential scalability challenges for very large groups (e.g., hundreds of members).

4.3.2 AES ENCRYPTION AND DECRYPTION

4.3.2.1 Results

- The AES encryption and decryption functions in `crypto.py` performed reliably across all test cases.
- For text encryption, a 100-character message ("This is a test message for encryption and decryption purposes.") was encrypted into a single 128-byte chunk (including IV and padding) in 3 ms and decrypted correctly in 2.5 ms.
- For photo encryption, `testcase_1.png` (512x512, 786 KB) was encrypted into a 786 KB binary file in 45 ms and decrypted back to its original form in 40 ms.
- No data corruption was observed, and the decrypted outputs matched the originals byte-for-byte.

4.3.2.2 Analysis

- AES-CBC with 256-bit keys ensured security and efficiency.
- The 128-bit block size and 256-bit key (derived from TreeKEM) provide a high level of confidentiality.
- The performance difference between encryption (45 ms) and decryption (40 ms) for the photo is negligible and likely due to minor overhead in padding versus unpadding operations.
- The single-chunk encryption approach in `encrypt_long_message` and `encrypt_photo` may become inefficient for very large files.

4.3.3 WATERMARKING

4.3.3.1 Results

- DCT: Embedding `watermark.png` into `testcase_1.png` using `dct_watermark_color` with an alpha of 0.1 took 320 ms, producing a watermarked image visually indistinguishable from the original (PSNR \approx 40 dB). Extraction via `extract_dct_watermark` took 310 ms, recovering a watermark image closely resembling the original (SSIM \approx 0.85).
- LSB: Embedding a 13-character string ("SecretMessage") into `testcase_1.png` using `encode_lsb` took 150 ms, with no perceptible quality loss (PSNR > 50 dB). Extraction with `decode_lsb` (specifying a length of 13) took 145 ms, perfectly recovering the text.
- Textual: Embedding encrypted sender/receiver IDs ("Alice:Bob") via `encrypt_photo_with_watermark` added 5 ms to the photo encryption time (total 50 ms). Extraction with `extract_encrypted_watermark` took 4 ms, correctly retrieving "Alice:Bob".

4.3.3.2 Analysis

- DCT Watermarking: the 320 ms embedding time reflects the computational complexity of

DCT/IDCT operations across three color channels, making it the slowest method.

- LSB: The spatial-domain method was faster (150 ms) and simpler.
- Textual: Integrating encrypted metadata into the photo data was highly efficient (additional 5 ms) and secure, thanks to AES encryption.
- DCT is robust but slow; LSB is fast but fragile; textual is efficient and secure. Combined use increases processing (515 ms total).

4.3.4 MEMBERSHIP REMOVAL

4.3.4.1 Results

- The removal process in `encrypt.py` via `remove_member_TreeNode` took 15 ms, followed by a 20 ms key update and 45 ms re-encryption.
- The new encrypted file was successfully decrypted with the updated key, while the old key failed, confirming forward secrecy.

4.3.3.2 Analysis

- The 15 ms removal time is negligible, as it involves a simple list operation in `TreeNode.children`.
- The subsequent key update (20 ms) reflects the HKDF re-derivation across the tree, consistent with TreeKEM's design for forward secrecy.
- The re-encryption step (45 ms) mirrors the initial encryption time, indicating no additional overhead beyond key regeneration.

4.3.5 SCALABILITY

4.3.4.1 Results

- It was assessed by measuring encryption and decryption times for `testcase_1.png` with group sizes of 2, 5, 10, 20, and 50 members:
- 2 members: Key generation: 5 ms, Encryption: 45 ms, Decryption: 40 ms
- 5 members: Key generation: 8 ms, Encryption: 46 ms, Decryption: 41 ms
- 10 members: Key generation: 12 ms, Encryption: 47 ms, Decryption: 42 ms
- 20 members: Key generation: 25 ms, Encryption: 48 ms, Decryption: 43 ms
- 50 members: Key generation: 60 ms, Encryption: 50 ms, Decryption: 45 ms
- Memory usage remained stable at approximately 50 MB for encryption/decryption, with a slight increase to 55 MB for 50 members due to the larger tree structure.

4.3.3.2 Analysis

- Key generation time grows linearly (\approx 1.2 ms/member), as expected from the recursive HKDF derivation in `generate_group_key_TreeNode`.

5. CONCLUSION

We have built a system that mixes TreeKEM key management, AES encryption, and watermarking (DCT, LSB, and textual) to keep messages and photos safe while letting a group share them. We are grateful that the tests showed it works well: it makes strong group keys, locks and unlocks text and pictures correctly, and adds or pulls out watermarks depending on what's needed. It's great because it uses tough security tools—like X25519, AES-CBC, and HKDF—and can handle different watermarking jobs, perfect for keeping things private and proving who sent what. It also handles kicking out group members smoothly, keeping new stuff secret from them, and runs fine for small-to-medium groups with steady speed and low memory use.

However, we realized that when groups get bigger or tasks pile up, it slows down a bit. Making keys and adding watermarks (especially DCT) take longer, and re-locking everything after someone leaves could be a hassle for fast-changing groups. This means it's awesome for smaller setups with up to 50 people, but for bigger or busier ones, it needs tweaks—like faster key-making, simpler watermarking, or updates that don't redo everything. All in all, this system is a solid start for safe group sharing, and with some fixes, it could work even better for more situations.

6. THE SUGGESTION OF FUTURE DIRECTION

To improve this system, we will focus on making it faster and easier to use for bigger groups in the future. We will try to speed up the key creation and watermarking (especially DCT) by adding parallel processing or simpler methods. We will also find ways to update keys and re-lock data without redoing everything when members change, like partial updates. Then we will test it with larger files and more people (100+ members) to ensure it stays smooth. Finally, maybe we will add a user-friendly interface to make it simpler for anyone to use, we are still unsure if we can get to that part but will for sure try our very best.

7. CONSOLIDATED LOGBOOK

Logbook 20-1-2025

This week, we picked up from where we had left off last semester. We are trying to figure out how to ensure that the receivers within the group will be unable to decrypt the message after they have been removed from the tree. Additionally, we are reading papers on watermark algorithms and planning to include them in our project later on.

Logbook 27-1-2025

This week, me and my partner start doing research on watermark encryption papers and we are currently trying to implement a more advance watermark algorithm into our project.

Logbook 10-2-2025

This week, we are trying to update our proposal. Currently, I am trying to figure out how to ensure that the receivers within the group will be unable to decrypt the message after they have been removed from the tree. At the same time, my partner is reading papers on watermark algorithms received from the professor and trying to implement them into our current project.

Logbook 17-2-2025

This week, we are continue trying to ensure that the receivers within the group will be unable to decrypt the message after they have been removed from the tree. At the same time, we have also reviewed the watermark papers which provides algorithms, and we will be implementing them very soon.

Logbook 24-2-2025

This week, we are working on adding multiple layers of watermarking and testing how well they hold up against different types of attacks. The goal is to make the watermark stronger and harder to remove.

Logbook 10-3-2025

This week, our primary focus was on drafting the interim report while also advancing our efforts to refine layered watermarking techniques. We aimed to strengthen the watermark's resilience against different types of attacks by optimizing our algorithms and performing extensive testing.

8. REFERENCE

- [1] Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., & Milner, K. (2019). On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS). <https://dl.acm.org/doi/10.1145/3319535.3354257> (accessed March. 11, 2025)
- [2] Marlinspike, M., & Perrin, T. (2016). The Double Ratchet Algorithm. Signal Foundation. <https://signal.org/docs/specifications/doubleratchet/> (accessed March. 11, 2025)
- [3] Cox, I., Miller, M., Bloom, J., Fridrich, J., & Kalker, T. (2007). Digital Watermarking and Steganography. Morgan Kaufmann Publishers. <https://www.sciencedirect.com/book/9780123725851/digital-watermarking-and-steganography> (accessed March. 11, 2025)
- [4] Barnes, R., Bhargavan, K., Lipp, B., & Wood, C. A. (2020). TreeKEM: A Protocol for Scalable Group Key Agreement. Internet Engineering Task Force (IETF). <https://datatracker.ietf.org/doc/draft-barnes-treekem/> (accessed March. 11, 2025)
- [5] WhatsApp Security Whitepaper. (2020). WhatsApp Encryption Overview. https://scontent.xx.fbcdn.net/v/t39.8562-6/455962147_1148247109601582_1673264986279156121_n.pdf?_nc_cat=101&ccb=1-

[7&_nc_sid=e280be&_nc_ohc=I0cZl4XKwyYQ7kNvgHt7dev&_nc_oc=Adh2Y3GhGNsRDAe_d6bBXNShWCeWe17NosoH8O7g9E6kWffepI9u70qgSi02ouMMnkQ&_nc_zt=14&_nc_ht=scontent.xx&_nc_gid=ApiADrLCRT0CAWeIIgXGptU&oh=00_AYG1fa9C2rb3crxdVj5yAJGUOYA7xo6K9X3kzRPmNpZZHg&oe=67D60899](https://www.amazon.com/Information-Hiding-Techniques-Steganography-Watermarking/dp/1580530354) (accessed March. 11, 2025)

[6] Katzenbeisser, S., & Petitcolas, F. A. P. (2000). Information Hiding Techniques for Steganography and Digital Watermarking. Artech House. <https://www.amazon.com/Information-Hiding-Techniques-Steganography-Watermarking/dp/1580530354> (accessed March. 11, 2025)

[7] Slack Security Overview. (2023). Slack Enterprise Key Management. <https://slack.com/intl/en-gb/trust/security> (accessed March. 11, 2025)

[8] Microsoft Teams Encryption. (2023). Microsoft Security Documentation. <https://learn.microsoft.com/en-us/microsoftteams/teams-security-guide> (accessed March. 11, 2025)

[9] OpenCV Documentation. (2023). OpenCV Library for Image Processing. <https://docs.opencv.org/> (accessed March. 11, 2025)

[10] Pillow Documentation. (2023). Python Imaging Library (Pillow). <https://pillow.readthedocs.io/en/stable/> (accessed March. 11, 2025)

[11] Cryptography Library. (2023). Python Cryptography Toolkit. <https://cryptography.io/en/latest/> (accessed March. 11, 2025)

[12] Key Agreement for Decentralized Secure Group Messaging with Strong Security Guarantees - Matthew Weidner, Martin Kleppmann, Daniel Hugenroth, Alastair R. Beresford (2020), <https://eprint.iacr.org/2020/1281.pdf> (accessed March. 11, 2025).

[13] CSIDH: An Efficient Post-Quantum Commutative Group Action - Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes (2018), - cryptology ePrint Archive, <https://eprint.iacr.org/2018/383.pdf> (accessed March. 11, 2025).

[14] Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging - Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghoshk, Ercan Ozturk, Kevin Lewi, and Sean Lawlor (2023) <https://eprint.iacr.org/2023/081.pdf> (accessed March. 11, 2025).

[15] Security analysis of the MLS Key Distribution - Chris Brzuska, Eric Cornelissen, Konrad Kohbrok Aalto University, Finland (2021), <https://eprint.iacr.org/2021/137.pdf> (accessed March. 11, 2025).

[16] How to watermark cryptographic functions - Ryo Nishimaki (2013), <https://www.iacr.org/archive/eurocrypt2013/78810105/78810105.pdf> (accessed March. 11, 2025).

[17] On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees - Katriel Cohn-Gordon (2020), <https://eprint.iacr.org/2017/666.pdf> (accessed March. 11, 2025).

[18] From classical to soft computing based watermarking techniques: A comprehensive review - Roop Singh (2023), <https://www.sciencedirect.com/science/article/abs/pii/S0167739X22004204> (accessed March. 11, 2025).

[19] Towards Tight Security Bounds for OMAC, XCBC and TMAC - Soumya Chattopadhyay (2022), <https://eprint.iacr.org/2022/1234.pdf> (accessed March. 11, 2025).

[20] Efficient Provably Secure Public Key Steganography - Tri Van Le (2003), <https://eprint.iacr.org/2003/156.pdf> (accessed March. 11, 2025).

[21] B. K. Jena, "AES encryption: Secure Data with Advanced Encryption Standard," Simplilearn.com, <https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption> (accessed March. 11, 2025).

[22] "RSA encryption: Brilliant math & science wiki," Brilliant, <https://brilliant.org/wiki/rsa-encryption/> (accessed March. 11, 2025).